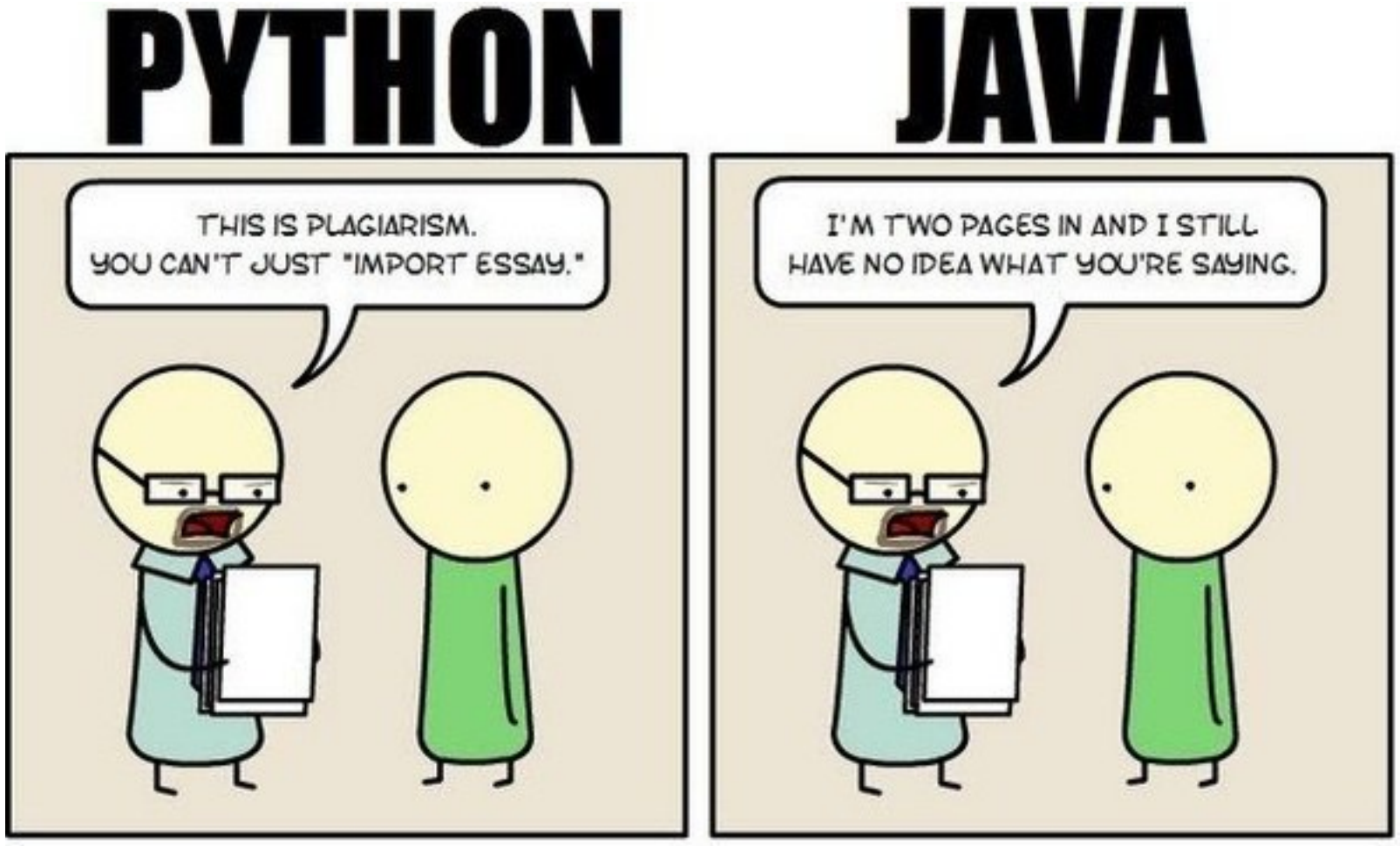
# Python 101

# Objectives

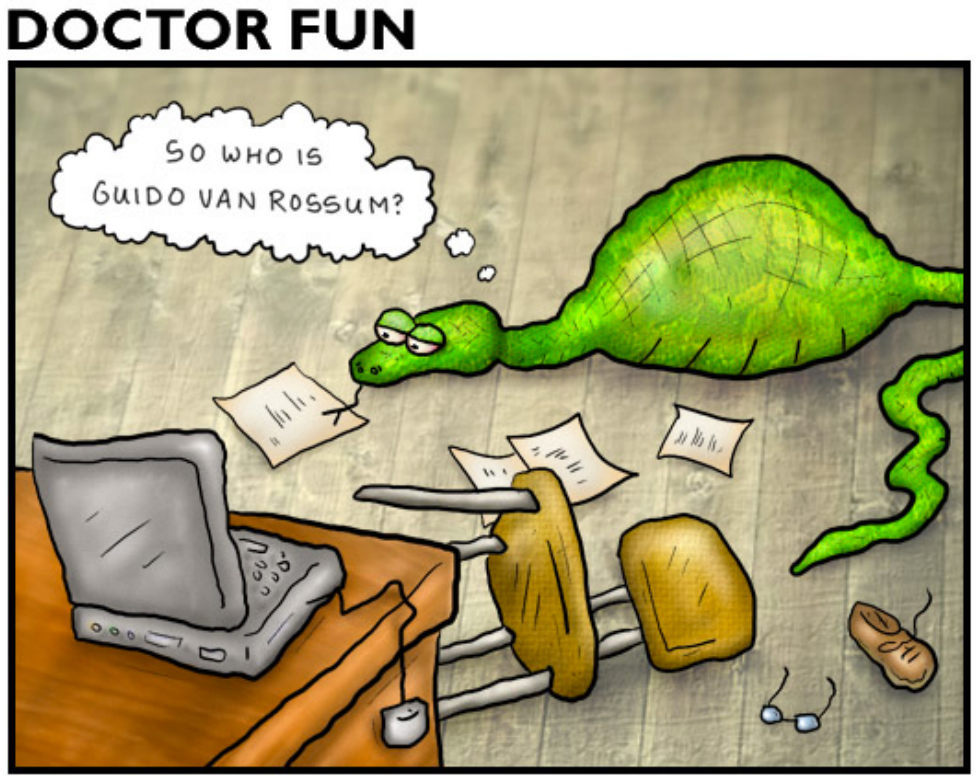
- Why python?
- Python2 vs Python3.
- Installing Python3.
- Objects and data structures.
- Working with numbers and variables.
- Working with Strings, indexing and slicing.
- Lists.
- Tuples.
- Dictionaries.
- Sets.
- Working with files.
- Booleans.
- Control flows (if statements).
- Loops.
- Working with Functions.
- Installing libraries.

# Why Python over Java?



# Why Python?

- Developed by **Guido Van Rossum** in 1991, in Netherlands.




- Advantages:
  - Easy to learn.
  - Typically involves less code.
  - Syntax is easier to read, focuses on indentation.
  - Utilized by every major technology company.
  - User friendly data structures.
  - Productivity and speed.
  - Popular with Data Scientist and ML.

- We are going to review a set of tools that…

**“are most recommended and useful with emerging technologies.”**

# Python 3 vs Python 2

- Choosing between Python 2 vs Python 3 **used to be a very difficult decision** for newcomers to the Python programming language.
- Many companies still had **legacy Python 2** code to be maintained.
- Python is going to **stop receiving** security **updates** in the year **2020**.
- Every major external Python package has been updated to support Python 3.
- **Python 3 is the future for Python!**

# Installing Python 3.

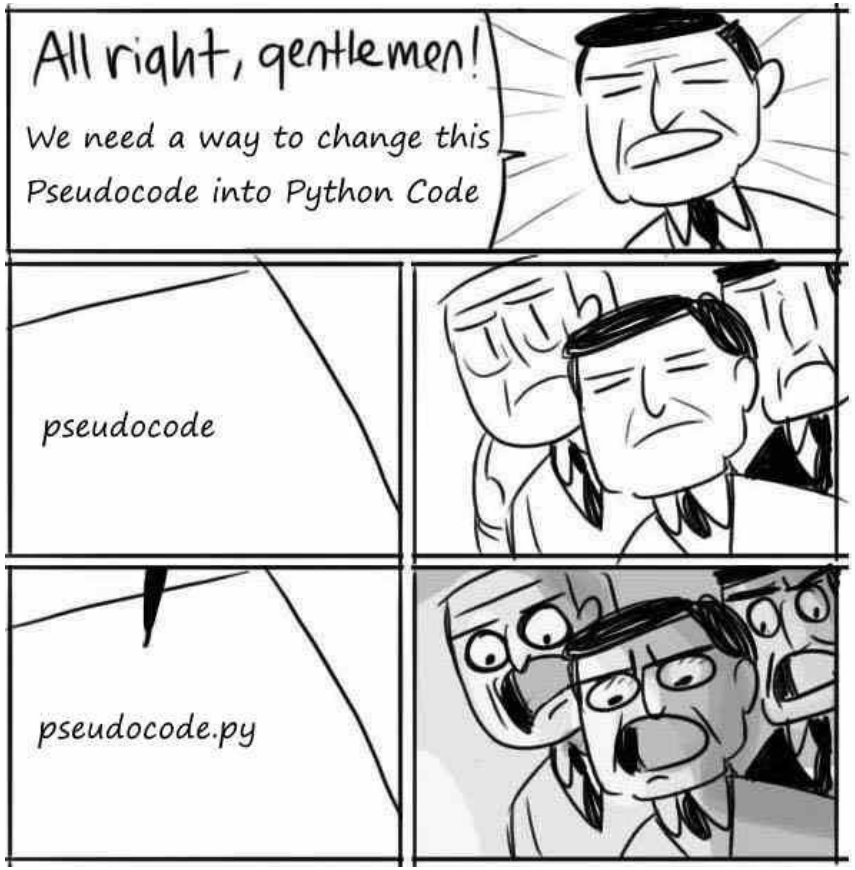
- Download link:
  - https://www.python.org/downloads/

# Working with Python

- We are going to install a Notebook environment.
  - Jupyter notebook.

  http://jupyter.readthedocs.io/en/latest/install.html#alternative-for-experienced-python-users-installing-jupyter-with-pip

# Python Hands - on



# Object and Data Structures

| Name | Type | Ordered | Example |
|---|---|---|---|
| Integer | int | - | Basic whole numbers, 1  2  10000 |
| Floating point | float | - | Numbers with decimal point: 2.3 4.6 200.0 |
| Strings | str | Yes | Sequence of characters: “Hi”  ”Mike” “1000” |
| Lists | list | Yes | Sequence of objects: [10,”hello”,200.3] |
| **Dictionaries** | **dict** | **No** | **Key:Value pairs: {“myKey”:”value”, “name”:”Frankie”}** |
| Tuples | tup | Yes | Immutable sequence of objects (10,”hello”,200.3) |
| Sets | set | No | Collection of unique objects: {“a”,”b”} |
| Booleans | bool | - | True or False |

# Python Hello World.

- Validate Python3 installation.
- My first python program with Python and Jupyter.

# Hands-on -1, Working with numbers

- There are two main number types:
  - **Integers.**
  - **Floating point.**

- Basic math with Python!

# Using variables

- Creating a variable in Python:

**my_dogs=2**

- Rules for variable names:
  - Names can´t start with numbers.
  - No spaces in the name, use "_" instead.
  - Names can´t contains special characters like:

:"´,<>/? | \()!@#$%^&*~-+

  - Use lowercase.
  - Avoid using words that have special meaning with Python like: "list" and "str".

# Using variables

- Uses **Dynamic Typing**, you can reassign variables to different data types.
- Python is flexible in assigning data types, not **"statically-typed".**
- **Example:**
  - **my_dogs=2**
  - **my_dogs="Two Dogs"**
  - **my_dogs=["Ginger","Glove"]**

This is Ok with Python.

- **Pros of Dynamic Typing:**
  - Very easy to work with
  - Faster development time
- **Cons:**
  - Prone to produce unexpected data types bugs.
  - You need to be aware of **type()**

# Working with Strings

- **Strings**, are ordered sequences of characters.
- **Strings** are immutable.
- **+** works for Strings concatenation.
- **#** to add comments.
- To work with Strings, we can use **indexing** and **slicing**.
- **Indexing**, uses [] notation after the String, to allow grabbing a single character from the String.
- **Example:**

  - **Characters:** **H E L L O**
  - **Index:** **0 1 2 3 4**
  - **Reverse Index:** **0 -4 -3 -2 -1**

# Working with Strings

- **Slicing**, allows you to grab a subsection of multiple characters, a "slice" of the String.
- Slicing, syntax:


**Start:** Numerical index for the slice start (zero based)

**Stop:** Index you will go up to (exclusive)

**Step:** is the size of the "jump"

# Activity with Strings

- From the next String, reverse it and then remove the first and last 3 characters.
- String:

**ewqbojdoogabt**

# String Properties and formatting methods

- Strings are Immutable.
- Strings are objects with interesting methods to use.
- Strings support 2 main methods to display variables.
  - **.format()**
  - **F-strings**

# Working with Lists

- Lists are ordered sequences, that can hold a variety of object types.
- Lists use brackets and commas to store and separate objects.

**[1,2,3,4,5]**

- Lists support **indexing** and **slicing**.

# Working with Dictionaries

- Unordered mappings for storing objects.
- Store **{Key:value}** pairs, this allow users to grab values quickly without needing an index location.
- Commonly used for **Json** objects representation.

**{'Key1':'Value1','Key2':'Value2'}**

# Working with Dictionaries

- Unordered mappings for storing objects.
- Store **{Key:value}** pairs, this allow users to grab values quickly without needing an index location.
- Commonly used for **Json** objects representation.

**{'Key1':'Value1','Key2':'Value2'}**

# Working with Tuples

- Similar to lists, but the key difference is that tuples are **immutable**.
- Once an element is inside a tuple, it can't be reassigned.
- Tuples use parenthesis and commas:

**(1,2,3)**

- Tuples are great to maintain data integrity, common used to avoid updating values in methods or larger pieces of code.

# Working with Sets

- Sets are unordered collections of **unique** elements.
- Only one element of the same object.

```
my_set = set()
```

# Working with Files

- Create files with Jupyter notebook.
- Python has different modes when dealing with files.
  - **mode='r'** is read only.
  - **mode='w'** is write only (will overwrite files of create new ones).
  - **mode='a'** is append only.
  - **mode='r+'** is reading and writing.
  - **mode='w+'** is writing and reading(Overwrites existing files or creates a new one ).

# Working with Booleans

- Booleans represent **True** or **False** statements.
- Important when dealing with control flow logic.

# Control Flows

- Control flows, use colons and indentation.
- Indentation is necessary for flow control.
- Basic syntax of it statement:

```
if some_condition:
        #execute code
else:
        #execute else code
```

# Control Flows

- Syntax of an if/else if statement:

```
if some_condition:
        #execute code
elif some_other_condition:
        #do something different
else:
        #execute else code
```

# Working with For loops

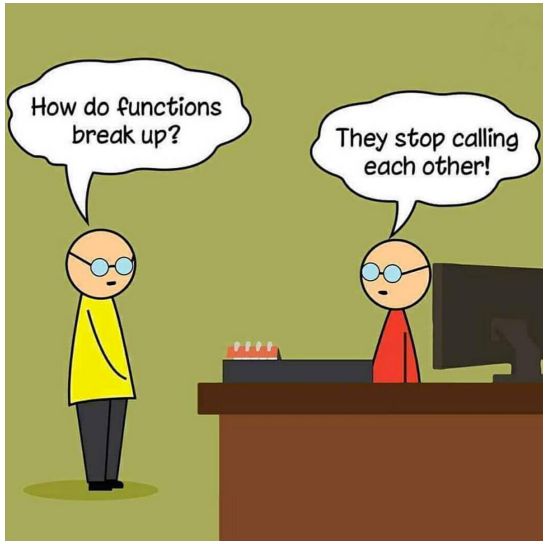
- For loops work with “Iterable” objects, that means, the ability to perform an action over every object in a collection.
- Example:

```
item_list= [1,2,3]
for item in item_list:
    print(item)
```

```
>>1
>>2
>>3
```

# Working with functions

- Creating re usable code is key par of becoming an effective programmer.
- **Functions** allow us to create blocks of code that can be easily executed many times, without needing to constantly rewrite the entire block of code.



# Functions syntax

```
#Using snake_case
def name_of_function(name):
    '''
    Function documentation
    '''
    print("Hello" + name)
    return name
```

# Using Pip

- **Pip** is a package management system used to install and manage software packages written in Python.

- Useful **pip3** commands:

  - **pip3 list**
  - **pip3 install XXXXX**
  - **pip3 uninstall XXXX**

# Now, you are allowed to say!

# Q & A